Build My Dapp

This report may include sensitive information about the customer's IT systems, intellectual property, potential security vulnerabilities, and methods to exploit them.

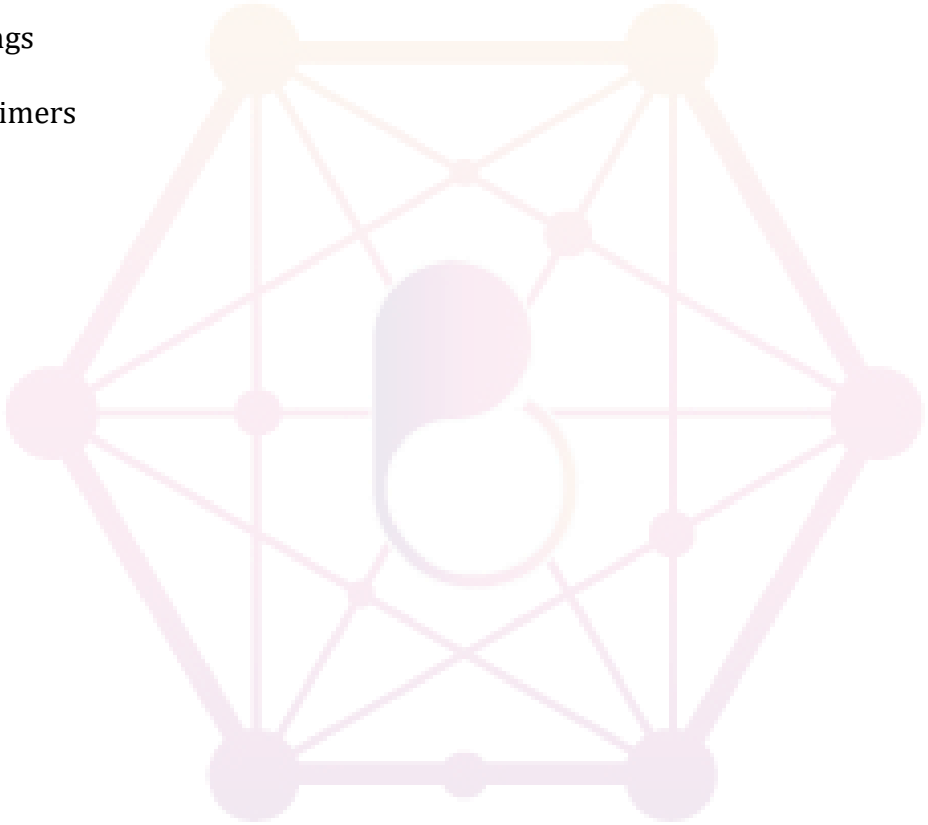
The report can be made public with prior approval from another party, but further publication does not require additional consent.

# Security analysis and review of Dalle Inu Mint Factory Smart contract code

**February 2023**

# Table of contents





## Introduction

The report presents the results of a security evaluation of the customer's smart contracts, which was carried out by BMD as part of a Smart Contract Code Review and Security Analysis project.

## Scope

The scope of the project is smart contracts in the repository:

**Link to deployed contract**:
https://etherscan.io/address/0xd42974fFF67de1EA3f414985bf9173808bea5Bf3
**Commit Hash**:
8a2b147806ba7952c046eca104d4882db71dfef2
**Technical Documentation**: Yes
**Integration and Unit Tests**: Yes
**Deployed Contracts Addresses**:
0xd42974fFF67de1EA3f414985bf9173808bea5Bf3

## Severity Definitions

| Risk Level | Description |
|---|---|
| Critical | Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations. |
| High | High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions. |
| Medium | Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations. |
| Low | Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution. |

## Executive Summary

### Documentation quality

The total Documentation Quality score is 10 out of 10. Functional requirements are provided and well-describes the product.

### Code quality

The total Code Quality score is 9 out of 10. Most of the code follows official language style guides. The unit tests are provided.

### Architecture quality

The architecture quality score is 10 out of 10. Code is separated to different contracts, following the single responsibility principle. Development environment is well set-up.

### Security score

As a result of the audit, the code contains 3 low severity issues. The security score is 10 out of 10.
All found issues are displayed in the “Findings” section.

### Summary

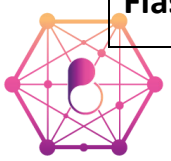
According to the assessment, the Customer's smart contract has the following score: 9.8.

## Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities:

| Item | Type | Description | Status |
|---|---|---|---|
| **Default Visibility** | SWC-100<br>SWC-108 | Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously. | **Passed** |
| **Integer Overflow and Underflow** | SWC-101 | If unchecked math is used, all math operations should be safe from overflows and underflows. | **Passed** |
| **Outdated Compiler Version** | SWC-102 | It is recommended to use a recent version of the Solidity compiler. | **Passed** |
| **Floating Pragma** | SWC-103 | Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. | **Passed** |
| **Unchecked Call Return Value** | SWC-104 | The return value of a message call should be checked. | **Passed** |
| **SELFDESTRUCT Instruction** | SWC-106 | The contract should not be self-destructible while it has funds belonging to users. | **Not Relevant** |
| **Check Effect Interaction** | SWC-107 | Check-Effect-Interaction pattern should be followed if the code performs ANY external call. | **Passed** |
| **Assert Violation** | SWC-110 | Properly functioning code should never reach a failing assert statement. | **Passed** |
| **Deprecated Solidity Functions** | SWC-111 | Deprecated built-in functions should never be used. | **Passed** |
| **Delegate call to Untrusted Callee** | SWC-112 | Delegate calls should only be allowed to trusted addresses. | **Not Relevant** |
| **DoS (Denial of Service)** | SWC-113<br>SWC-128 | Execution of the code should never be blocked by a specific contract state unless it is required. | **Passed** |
| **Race Conditions** | SWC-114 | Race Conditions and Transactions Order Dependency should not be possible. | **Passed** |

| | | | |
|---|---|---|---|
| **Authorization through tx.origin** | SWC-115 | tx.origin should not be used for authorization. | **Passed** |
| **Block values as a proxy for time** | SWC-116 | Block numbers should not be used for time calculations. | **Passed** |
| **Signature Unique Id** | SWC-117 SWC-121 SWC-122 EIP-155 | Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery | **Not Relevant** |
| **Shadowing State Variable** | SWC-119 | State variables should not be shadowed. | **Passed** |
| **Weak Sources of Randomness** | SWC-120 | Random values should never be generated from Chain Attributes or be predictable. | **Not Relevant** |
| **Incorrect Inheritance Order** | SWC-125 | When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. | **Passed** |
| **Calls Only to Trusted Addresses** | EEA-Level-2 SWC-126 | All external calls should be performed only to trusted addresses. | **Passed** |
| **Presence of unused variables** | SWC-131 | The code should not contain unused variables if this is not justified by design. | **Passed** |
| **EIP standards violation** | EIP | EIP standards should not be violated. | **Passed** |
| **Assets integrity** | Custom | Funds are protected and cannot be withdrawn without proper permissions. | **Passed** |
| **User Balances manipulation** | Custom | Contract owners or any other third party should not be able to access funds belonging to users. | **Passed** |
| **Data Consistency** | Custom | Smart contract data should be consistent all over the data flow. | **Passed** |
| **Flash loan Attack** | Custom | When working with exchange rates, they should be received from a trusted source | **Not** |



| | | | |
|---|---|---|---|
| | | and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used. | **Relevant** |
| **Token Supply manipulation** | Custom | Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer. | **Passed** |
| **Gas Limit and Loops** | Custom | Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit. | **Passed** |
| **Style guide violation** | Custom | Style guides and best practices should be followed. | **Passed** |
| **Requirements Compliance** | Custom | The code should be compliant with the requirements provided by the Customer. | **Passed** |
| **Environment Consistency** | Custom | The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code. | **Passed** |
| **Secure Oracles Usage** | Custom | The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles. | **Not Relevant** |
| **Tests Coverage** | Custom | The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested. | **Passed** |
| **Stable Imports** | Custom | The code should not reference draft contracts, that may be changed in the future. | **Passed** |

## Findings

- ### Critical
  No critical severity issues were found.

- ### High
  No high severity issues were found.

- ### Medium
  No medium severity issues were found.

- ### Low

**1. Public functions instead of external**

Some functions are declared as public, although they are not called internally in the related contract. Public function visibility consumes more Gas than external visibility.

**Recommendation**: Change public visibility with external.

**Status**: Fixed

**2. Missing event emitting**

Events for critical state changes should be emitted for tracking things off-chain.

**Recommendation**: It is recommended to add events in functions that change state.

**Status**: Fixed

**3. Floating pragma**

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

**Recommendation**: It is recommended to lock the pragma version whenever possible to avoid potential issues with deploying contracts using outdated compiler versions. It is also advisable to avoid using a "floating" pragma in the final deployment.

**Status**: Fixed

## Disclaimers

### BMD Disclaimer

The report provides an analysis of the smart contracts submitted for audit using the best industry practices at the time of the report. It includes details of any cybersecurity vulnerabilities and issues found in the source code. However, it does not guarantee that all vulnerabilities have been identified, and the code may not be secure. The report only covers the code reviewed and may not be relevant if the code is modified. It should not be considered a final and sufficient assessment of the code's utility, safety, or bug-free status. The consultant suggests conducting multiple independent audits and a public bug bounty program to ensure the security of smart contracts. The report is originally in English and the consultant is not responsible for the accuracy of translated versions.

### Technical Disclaimer

Smart contracts are executed on a block chain platform, but the platform, programming language, and other related software may have vulnerabilities that could lead to hacking. As a result, it is not possible for the consultant to ensure the complete security of the audited smart contracts.